CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; 00:1–19
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Analyzing the Impact of Communication Imbalance in High-Speed Networks

Gladys Utrera, Marisa Gil and Xavier Martorell

Computer Architecture Department. Jordi Girona 1-3, Universitat Politècnica de Catalunya, Barcelona, 08034, Spain

SUMMARY

In this work we analyze the communication load imbalance generated by irregular-data applications running in a multi-node cluster. Experimental approaches to diminish communication load imbalance are evaluated using a hybrid programming model MPI+OpenMP including certain optimizations like computation-communication overlap, issuing communications in parallel and a new proposal based on message fragmentation in order to take advantage of the eager-protocol. Performance results show that overlapped versions can obtain a great benefit of this optimization because it avoids switching to rendez-vous protocols. However, non-overlapped versions showed better performance than overlapped ones. To evaluate also the impact due to network latency, the work has been tested on two high-speed interconnection networks: Infiniband and 10Gigabit Ethernet. In this case, the optimizations in the non-overlapped miniFE benchmark reached and improvement up to 7% on Infiniband and 11% on 10Gigabit.

Received . . .

KEY WORDS: scalable computing; load balancing; high performance interconnection networks; MPI library; message size

1. INTRODUCTION

Many High-Performance Computing (HPC) applications exist today that range from models of the physical world to web search and graph clustering, which are composed of data structures that are sparse matrices and involve techniques such as linear algebra and graph algorithms. These irregular data-structures do not exploit block structures or symmetries, which gives rise to both computation and communication load unbalancing as well as affecting parallel scalability and performance in multi-node systems.

As regards the interconnection network, high-speed networks such as 10Gigabit Ethernet or InfiniBand that are widely used as a fast network and figure prominently in the Top500 [26], may exhibit irregular behavior when the amount of messages increases. This happens because most of the MPI (Message Passing Interface [20]) implementations usually uses a two-level protocol for point-to-point messages: the rendez-vous protocol for long messages and the eager protocol, which optimizes latency, for short messages. There is an eager-limit value to decide whether a message is large enough to move from eager to rendez-vous protocol [25], [18]. This protocol aims to prevent the memory overflow of communication buffers and begins the communication by sending a first control message to the receiver in order to synchronize the send-receive flow of data messages. Another situation where this protocol is used when the number of short messages sent to the receiver endangers memory buffers to overflow.

Correspondence to: Office D6-102, Jordi Girona 1-3, Universitat Politècnica de Catalunya, Barcelona, 08034, Spain
E-mail: gutrera@ac.upc.edu

Prepared using cpeauth.cls [Version: 2010/05/13 v3.00]

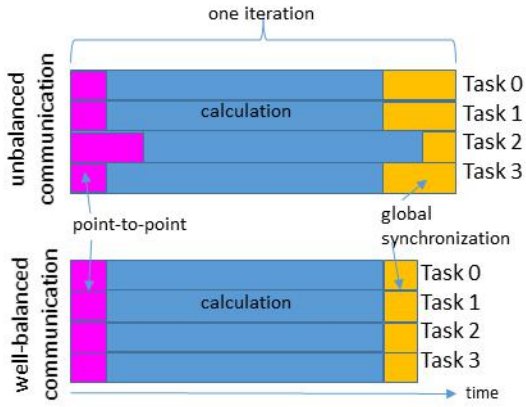

Figure 1. Example of the impact of load imbalance due to communication: a single iteration execution with communication load imbalance (top) and the corresponding well-balanced execution (bottom).

In this work, we focus on the specific load imbalance due to communication generated by these two factors: the communication library and the message size. Let's take a look at example in Figure 1 to explain the impact of communication imbalance in the overall application performance. In this example, one may see two executions, each one representing a single iteration in the execution of 4 tasks. Each execution is composed initially by point-to-point communication (light grey / red-pink color), then calculation (blue / dark grey color) and finally a global synchronization (gray / orange color). The calculation part is well-balanced. However, the execution on top figure is unbalanced because of the initial communication phase. One of these tasks (task 2) has a longer point-to-point communication phase than the rest, so when each of the tasks arrive at the global synchronization phase, at the end of the iteration, they have to wait for the slowest one, consequently requiring a bigger synchronization phase than the well-balanced case (bottom figure).

Therefore, the communication phase may participate in the application imbalance, even in applications that have a well-balanced calculation part across tasks. Such imbalance is propagated through the synchronization phase to all the tasks, affecting the overall performance of the application.

We analyze two main factors that produce this imbalance: the communication library and the different number and size of messages during the communication phase generated specially by irregular applications.

Traditionally communication imbalance is hidden by overlapping communication and calculation [1], [8]. In a previous work, we analyzed the performance of overlapping algorithms implemented using fork-join and task based implementations. We concluded that the MPI protocols should be adjusted in order to obtain an acceptable performance.

In this work, we look at the latency impact by running the experiments in two different interconnection networks: Infiniband and 10 Gigabit Ethernet, and we explore new ways of elongate the "eager-limit" as much as possible as it is identified as a critical bottleneck. Part of the Infiniband results were presented in a previous work [27].

In order to get a deeper understanding of the execution internals, we begin the analysis using two synthetic applications that allow us to control the factors we want to explore and categorize the effects. Then, we choose miniFE, a Mantevo suite [19] miniapp as our case study. This benchmark allows us to concentrate on the performance of the Sparse Matrix Vector multiplication (SpMV), which is widely used in linear algebra and graph algorithms and a good representative of HPC applications that work on irregular structures. The miniFE miniapp benchmark, it is similar to High Performance Computing Conjugate Gradient (HPCCG) [10] though it provides a much more complete vertical coverage of the steps in this class of applications. Like HPCCG, miniFE is intended to be the best approximation to an unstructured implicit finite element or finite volume application, but in 8,000 lines or less. The HPCCG Benchmark project is an effort to create a more relevant metric for ranking HPC

DOI: 10.1002/cpe

systems than the High Performance Linpack (HPL)[11] benchmark, which is currently used by the Top500 benchmark.

The contributions of this work are as follows:

- Investigate comprehensive different sources of load imbalance due to communication when executing applications with different communication degree
- Analyze of latency hiding strategies using Infiniband and 10Gigabit Ethernet interconnection networks
- Evaluate alternatives to reduce such load imbalance: a) tuning the eager-limit; b) parallelization of the data transfer; and a new approach: c) split messages so that they fit the eager-limit size
- Compare high speed interconnection networks when tuning applications to alleviate communication load imbalance

We observe in our experiments that non-overlapped versions perform better than overlapped versions. However, when executing on 10Gigabit where latency is a performance factor, the overlapped versions show better performance than non-overlapped ones, if and only if, message sizes fit the eager-limit. With respect to the proposed optimizations, in miniFE, less than spectacular, performance improvements range from 7% on Infiniband to 11% on 10Gigabit for the non-overlapped versions and message sizes up to 512 Kilobytes (Kb).

Our best improvements in performance on Infiniband, come from the overlapped version, because of the efforts to avoid switching unnecessarily to rendez-vous protocol. The improvements in performance obtained on 10Gigabit interconnection network, come also from the reduction of the total message passing latencies. In this case, the latencies are high enough to justify the overhead of the optimizations for algorithms without overlapping, which is not the case on Infiniband.

Finally, experimental results show that the communication network should be taken into account when deciding the parallelism structure: with proper optimization settings, executions on interconnection network 10Gigabit can sometimes reach as much performance as an InfiniBand network.

2. RELATED WORK

The load imbalance problem deserves special attention; as stated in [9]: “Load balance is one of the most important factors limiting scalability in charge scale parallel computation. Hence understanding its impact and source is an essential step in improving application performance”. This has given rise to an extensive literature devoted to this issue; for example, in [6] the authors analyze the impact of this load imbalance on the simulations of irregular and dynamic domains, in the case of real applications in astrophysics. Their work concludes that for this type of application, over 12.5% of the time is spent in wait states and 70% of this percentage comes from propagation of the wait state itself.

In [5], the authors present a sophisticated methodology to solve efficiently the bottleneck caused by the imbalance by studying alternatives based on the overlap of communication and computation to reduce this imbalance. In this work, authors propose an interesting method for calculating certain performance indicators based on the critical-path in order to provide insights into load balance cases that they combine with the performance information obtained from the Scalasca tool. This methodology is inapplicable in our case, since our goal is to solve the bottleneck during the runtime. Furthermore, the load imbalance does not only depend on the implementation of the application, but also on the pattern by which MPI messages are delivered (i.e. the order and time of arrival at the target process) which is unpredictable at runtime. The time of arrival is what potentially generates the communication load imbalance.

Having scalability as one of the main objectives because of Exascale computing, collective operations become a recurrent point of study. In a recent work [22], the authors study collective

DOI: 10.1002/cpe

operations such as the reduction and broadcast algorithm as well as proposing a dynamic leader node at the inception of a collective. The latency differences between intranode and internode communication must be taken into account. The strategy proposed in this work is not applicable, because the type of imbalance addressed is based on the platform rather than on the application itself.

Several works exist that deal with the eager protocol. For example, in [6] and [29], the authors show the benefits of choosing a protocol according to the network and the specific machine in which the application is running. In [7], a comparison is made of both MPI protocols, eager and rendez-vous, and the conclusion is that the eager protocol outperforms the standard rendez-vous if a large percentage of receive operations are pre-posted. In [24] the authors determine that, depending on the particular MPI implementation, non-blocking primitives can act as blocking primitives when the eager-limit is exceeded. They propose strategies to be used in conjunction with the non-blocking primitives to reduce processor synchronization overheads instead of addressing the issue of reaching the eager-limit itself, as proposed in this work. In [16] the authors evaluate the performance of the rendez-vous protocol in order to better understand message-passing protocols and to improve the overall distributed application performance. In [8], the authors propose three different alternatives to the traditional rendez-vous protocol on Infiniband based on the message size. The evaluation is done using the NAS Benchmarks up to 16 nodes and for small classes (A,B,C), using dense matrices instead of sparse matrices as is the case of this work.

Focusing on application implementation issues, a principal feature of our proposal is that we address the application with the original software as it is given. This means, for example, we maintain the same partitioning software (ParMeTis) and the same sparse matrix format (in this case, Compressed Sparse Row (CSR)). ParMeTis is the parallel graph-partitioning library used in this work. It is one of the most used for MPI irregular structures, using CSR format to work with sparse matrices, though poor performance scalability has been detected when the number of processors increases [17]. However, works exist that propose several partitioning algorithms to deal with computation imbalance, such as [14] or [23]. The authors in [6] propose a new partitioning algorithm to distribute the work, which results in fewer communication messages required than a traditional 1d partitioning algorithm. Other research works such as [15] have been conducted to propose new partitioning algorithms with the aim of improving performance.

Finally, there are works like [16], where the authors propose a tuning of SpMV although the work focuses on memory performance rather than communication. They achieved an improvement of up to 64%.

3. DESCRIPTION OF THE BENCHMARKS USED FOR THE ANALYSIS

First we introduce the description of our synthetic benchmarks (*noOverlapToy* and *overlapToy* applications) used for the preliminary analysis. In this way, we can isolate characteristics such as message number and sizes, as well as communication degree. Then, we proceed with the presentation of miniFE benchmark to validate our results. At the end of the section there is a description of the explored MPI protocols used in this work.

3.1. Synthetic benchmarks

The synthetic benchmark used is regular and iterative, and at each loop iteration we can distinguish three phases: point-to-point communication, calculation, group-communication. We have developed two versions of the benchmark: *noOverlapToy* and *overlapToy*.

The *noOverlapToy* with the source code shown in Figure 4, follows a scheme like the one shown in Figure 2. There one can observe a synchronous communication scheme, with 4 tasks in this example where task 0 is the master and executes the communication phase, which consists in an *MPI_Irecv, an MPI_Send* and finally an *MPI_Waitall* until message reception finishes. Then, the calculation phase begins. This version has no overlapping of computation and communication.

DOI: 10.1002/cpe

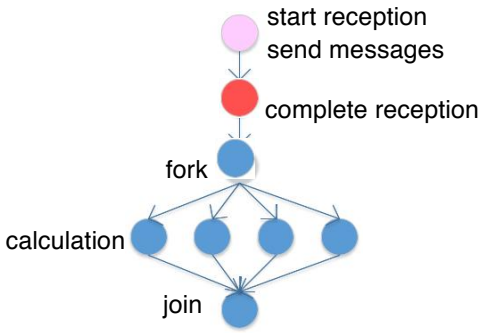

Figure 2. Scheme of MPI+fork-join based parallelism without overlap implementation.

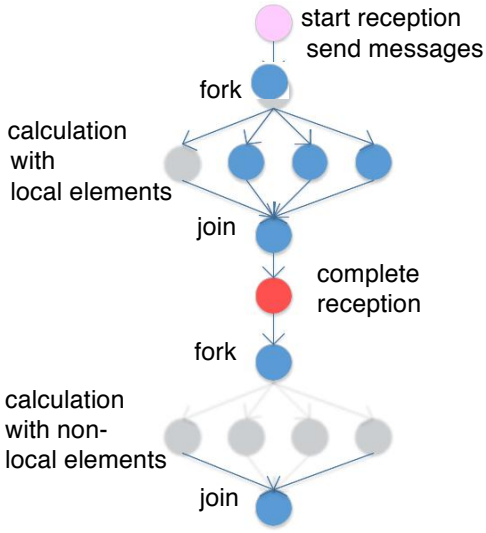

Figure 3. Scheme of MPI+fork-join based parallelism with overlap implementation.

The result of overlapping of some of the calculations with the communication part may be observed in Figure 3. First the posting and sending of the message is performed; the first part of the calculation is then carried out, and after the completion of the message reception, the second part of the calculation is performed. This scheme is used to implement the other synthetic benchmark, the *overlapToy*. The source code can be seen in Figure 5.

The communication degree (number of target processes), the number of messages, as well as the size of the messages are parametrized. The message size parameter express an upper limit. Communication imbalance due to data distribution is achieved by assigning randomly different sizes to messages and up to this parameter. In this way, the larger the message size, the bigger the communication imbalance. Once established, we repeat the same message sizes and pattern across all the iterations.

Table I summarizes the percentage of time spent in communication during execution of the parallel implementations of the synthetic benchmarks. Both, *noOverlapToy* and *overlapToy*, were run using 1 Megabyte (Mb) message sizes, 16 MPI tasks and every task send/receive a message to every other task.

DOI: 10.1002/cpe

```
for (loop=0; loop<MAXLOOP; loop++) {
    for (k=0; k<recvcount; k++) {
        i = recvlist[k];       // index of target process
        bufsize = from[i]; // message size
       MPI_Irecv (array_recvbufs[i], bufsize, MPI_CHAR, i, stag,  MPI_COMM_WORLD, &requests[k]);
    }
    for (k=0; k<sendcount; k++) {
        i = sendlist[k];      // index of source process
        bufsize = to[i];      // message size
        MPI_Send (array_sendbufs[i], bufsize, MPI_CHAR, i, rtag, MPI_COMM_WORLD);
    }
   if (recvcount>0) MPI_Waitall(recvcount, requests, statuses); // reception completion
        Calculation ();
        MPI_Allreduce (sendbuf, recvbuf, k, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
   }
}
```

Figure 4. *noOverlapToy* source code of main loop

```
for (loop=0; loop<MAXLOOP; loop++) {
   for (k=0; k<recvcount; k++) {
       i = recvlist[k];   // index of source process
       bufsize = from[i]; // message size
       MPI_Irecv (array_recvbufs[i], bufsize, MPI_CHAR, i, tag, MPI_COMM_WORLD, &requests[k]);
   }
   for (k=0; k<sendcount; k++) {
       i = sendlist[k];   // index of target process
       bufsize = to[i];   // message size
       MPI_Send(array_sendbufs[i], bufsize, MPI_CHAR, i, tag, MPI_COMM_WORLD);
   }
   Calculation_1();  // 90% of calculation
   if (recvcount>0) MPI_Waitall(recvcount, requests, statuses); // reception completion
   Calculation_2();  // 10% of calculation
   MPI_Allreduce(sendbuf, recvbuf, k, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
}
```

Figure 5. *OverlapToy* source code of main loop

Table I. % of execution time spent in communication when having point-to-point communication with 1 Mb messages sizes from/to every of the 16 MPI tasks running on 10Gigabit and Infiniband interconnection networks

benchmark	10Gigabit	Infiniband
noOverlapToy	33%	2%
overlapToy	68%	69%

3.2. miniFE miniapplication

We have chosen the miniFE performance benchmark, version 2.0 for our evaluation purposes. This mini-application belongs to the Mantevo project and encapsulates the main characteristics of an implicit finite element application. MiniFE solves a linear-system using unpreconditioned conjugate gradients (CG) with the kernel *SpMV*, which carries out the *SpMV* and represents the most significant fraction of the miniFE total run-time. The shape corresponding to these linear-systems is a multi-diagonal sparse matrix.

The sparse matrix storage format used in our implementation of the miniFE benchmark is Compressed Sparse Row (CSR), as it supports efficient access and matrix operations. The work partitioning among processes is done through the ParMetis library guaranteeing a well-balanced application. However, the CSR format does not exploit block structures or symmetries, which leads to communication load unbalancing.

Table II summarizes the percentage of time spent in communication during execution of the parallel implementations of miniFE using input matrix size of 1048x1048x512 and 64 MPI tasks.

DOI: 10.1002/cpe

We used for evaluations two versions of the benchmark: miniFE without communication/compu-tation overlapping and miniFE with communication/computation overlapping. The scheme of both versions is the same used to implement the synthetic benchmarks (figures 2 and 3 respectively). In the rest of the paper we refer to *miniFEnoOverlap* to the non-overlapped implementation of miniFE, and *miniFEoverlap*, to the overlapped implementation of miniFE. Each MPI task was run on a different node.

One can observe that we can expect to improve the performance on both benchmarks on 10Gigabit and just with the overlapped versions of the benchmarks on Infiniband.

Table II. % of execution time spent in communication when using a an input matrix size of 1048x1048x512 and 64 tasks, running on 10Gigabit and Infiniband interconnection networks

benchmark	10Gigabit	Infiniband
miniFE noOverlap	45%	10%
miniFE overlap	59%	64%

3.3. MPI protocols

The MPI eager protocol makes a copy of the message between sender and receiver, thereby improving management overhead as the sender process does not need any acknowledgement from the receiver to send the message. This is possible with small messages: messages under “eager-limit” size.

To prevent message size from compromising the memory space, for messages with size over the eager-limit, the MPI protocol establishes a synchronization between sender and receiver: the sender sends a first message indicating the intention to begin the communication and the size of the message; the receiver agrees and then communication begins.

This is the rendez-vous protocol [25], [18] that requires extra time due to the control message exchanged between the two nodes. Although it is not an MPI standard, almost all MPI implementations offer an optimization based on message size.

A characteristic of high-speed communication networks is their ability to change the protocol from eager to rendez-vous, even if the messages do not reach the default eager-limit [24], [28]. They do this in order to prevent memory from running out when the buffers are full, which may be done because the message flow is increasing. This behaviour may seriously hinder other factors, and for this reason one of our evaluations includes the tuning of the eager-limit value.

4. ANALIZING THE COMMUNICATION PHASE AND POTENTIAL SOURCES OF COMMUNICATION IMBALANCE

The latency of a message (the total time a message takes to go from source to destination) depends on the interconnection network and the message size. One can observe that effect in Table III. The table shows the results extracted from the execution of the Ping-pong Intel MPI Benchmark [12]. This benchmark repeats 1000 times for each message size the execution of point-to-point operations between two processes (*MPI_Send*/*MPI_Recv*) measuring and averaging the time spent in these operations. Recall that these results where taken from two executions, with exactly the same platform but different interconnection networks.

The executions of *noOverlapToy* and *overlapToy* synthetic benchmarks with different message sizes and communication degree (number of target processes), running on 10Gigabit and Infiniband interconnection networks, demonstrate what we observed with the IMB benchmark [12]: execution times increment with the message size and the communication degree as expected, especially on 10Gigabit.

However, the interesting results arise when taking a closer look at the execution of the *overlapToy* benchmark on both interconnection networks. Although Infiniband has smaller latencies than

DOI: 10.1002/cpe

Table III. IMB Point-to-point benchmark execution for different message sizes

Message size (kbytes)	10Gigabit (msecs)	Infiniband (msecs)
32	576.8	12.0
64	1193.5	19.0
128	1569.9	33.2
256	2695.9	60.5
512	4944.3	116.2
1024	9069.9	227.4
2048	17694.0	450.5
4096	34568.8	897.3

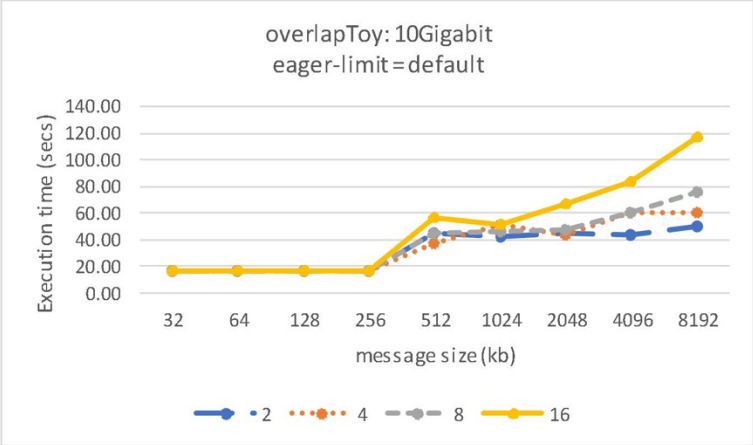

Figure 6. Execution times of *overlapToy* with 16 MPI tasks varying message size (between 32Kb and 8Mb) and number of communicating tasks (between 2 and 16) running on 16 nodes using 10Gigabit interconnection network and default eager-limit (256Kb)

10Gigabit, the performance is very poor with respect to 10Gigabit. This happens because of the automatic change of protocol from eager to rendez-vous even the eager-limit is not reached, when the message flow increases (see section 3.3). The maximum allowed value of this limit on 10Gigabit is higher than on Infiniband.

4.1. Eager-limit

Figures 6 and 7 show the impact of the eager-limit on the execution times of the *overlapToy* on both interconnection networks when varying the message size and the communication degree. Even though the eager limit is set to 512Kb, on Infiniband the change of protocol to rendez-vous is automatically applied before reaching that limit.

The reason why performance degrades with the overlapped version, is related with the implementation of the rendez-vous protocol with a two-phase communication protocol.

Figures 8 and 9 show a zoom of 350 ms of the execution of *overlapToy*, with 4 MPI processes visualized with Paraver tool [4]. The benchmark is running on 4 nodes using Infiniband interconnection network. Each horizontal bar corresponds to the execution of an MPI process allocated on a node. The lines (yellow/light grey) crossing the horizontal bars represent the communication (*MPI_Send*). The different colors represent different states of the execution and the main ones are as follows: blue/black is running outside MPI (calculation part), orange/ grey is global synchronization (*MPI_Allreduce*), pink/dark grey is *MPI_Send*, and red are wait states (*MPI_Wait*).

DOI: 10.1002/cpe

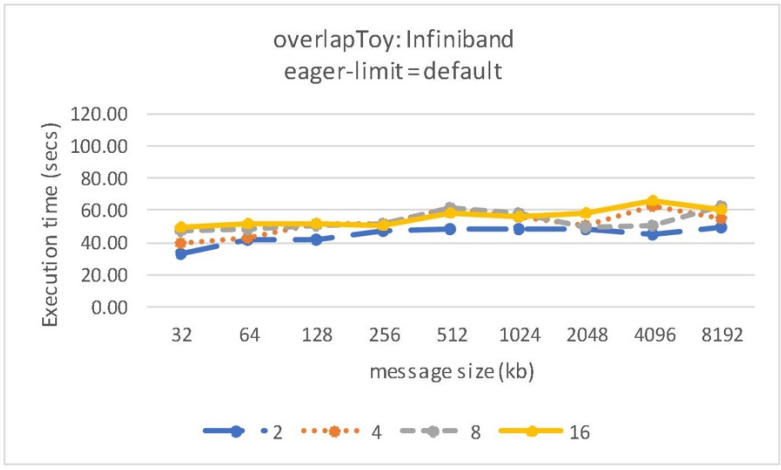

Figure 7. Execution times of *overlapToy* with 16 MPI tasks varying message size (between 32Kb and 8Mb) and number of communicating tasks (between 2 and 16) running on 16 nodes using Infiniband interconnection network and default eager-limit (256Kb)

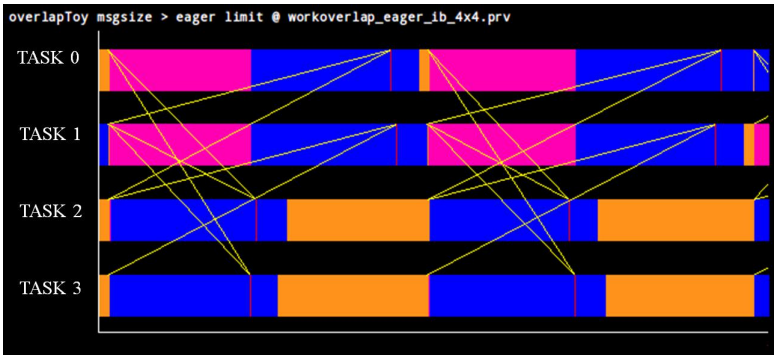

Figure 8. Paraver visualization of *overlapToy* with 4 MPI processes, 128Kbytes message sizes and eager-limit set to 64Kbytes

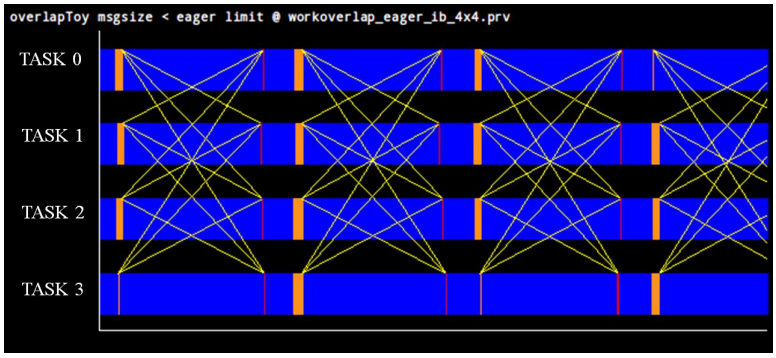

Figure 9. Paraver visualization of *overlapToy* with 4 MPI processes, 128Kbytes message sizes and eager-limit set to 128Kbytes

In order to understand the execution flow of a single iteration one may consult the source code shown in Figure 5. First we encounter the communication phase which we can observe in the figures that has different time durations, then comes the calculation part with fixed time duration, and finally each iteration ends with a global synchronization (*MPI_Allreduce*). This operation evidences any load imbalance generated inside each iteration. All the MPI tasks are able to continue execution just

DOI: 10.1002/cpe

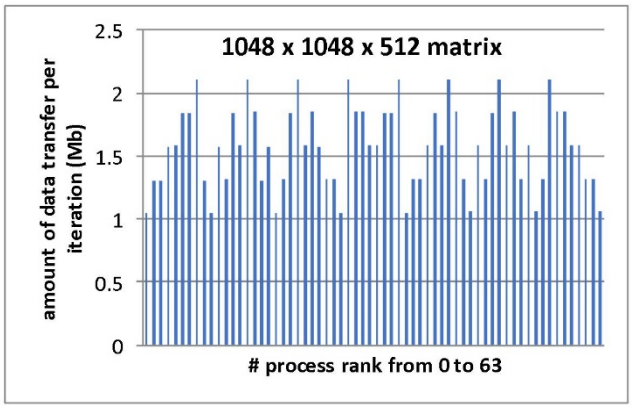

Figure 10. Data sent in Megabytes (Mb) each SpMV execution by each process on 64 nodes when working on a matrix of size 1048x1048x512.

after all of them have arrived at the synchronization point. Consequently, the duration of a given iteration is equal to the longest process time iteration.

In Figure 8, the global synchronization state is very remarkable for tasks 2 and 3 and almost indistinguishable for tasks 0 and 1. This happens because of load imbalance due to communication phases. One may observe that the communication phase (pink/dark grey state) occupies as much time as the calculation phase (blue/black) at each loop iteration. This happens as a result of the MP I Send operation, which is blocked until the target process actually performs the wait for completion of the reception of messages. This should not happen, as the corresponding MP I Irecv is already posted. However, as mentioned in Section 3.3, the MPI protocol of messages on Infiniband may sometimes switch from eager to rendez-vous when there is not enough space to allocate all the receiving messages, even though the eager-limit is not reached [24].

When adjusting the eager-limit specifically to the application being executed, that is setting its value greater or equal to the biggest message size, we can obtain an execution like shown in Figure 9 (message sizes smaller than or equal to the eager-limit).

However, the eager-limit on our platform cannot be pushed beyond 128Kb. Consequently, for applications having bigger message sizes this optimization technique is not applicable. This impact can be appreciated in the execution times of Figure 7 where even though the eager-limit is set to 512Kb, executions with message sizes over 128Kb but below 512Kb degrade performance significantly.

For all these reasons, it is necessary to investigate another way to reduce the communication phase when dealing with message sizes over the allowed maximum eager-limit.

4.2. Data transfer: number and size of messages

Irregular applications like the ones that work with sparse matrix, may generate load imbalance during the communication phase, because of the exchange of different number and size of messages.

Figures 10 and 11 show the total amount of data in megabytes sent per process and the total number of messages sent per process respectively, for one iteration execution of miniFE benchmark with 64 MPI tasks and an input matrix of size 1048x1048x512.

One may observe that every task send different amount of data and different number of messages. The total load imbalance is about 25% in both cases (amount of data and number of messages).

DOI: 10.1002/cpe

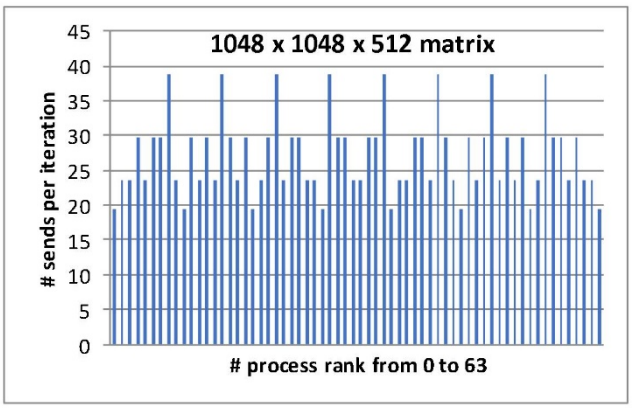

Figure 11. Number of messages sent at each SpMV execution by each process on 64 nodes when working on a matrix of size 1048x1048x512.

```
#pragma omp parallel for schedule (dynamic)
for (int k=0; k<sendcount; k++) {
   int i = sendlist[k];   // index of target process
   unsigned long long bufsize = to[i];   // message size
   MPI_Send(array_sendbufs[i], bufsize, MPI_CHAR, i, tag, MPI_COMM_WORLD);
}
```

Figure 12. MPI/OpenMP [21] code for the implementation of the parallel data transfer.

5. PROPOSALS TO DIMINISH LOAD IMBALANCE GENERATED DURING THE COMMUNICATION PHASE

Previously in Section 4.1 we have been investigating the effect of the eager-limit with different interconnection networks, communication degree and message sizes. In addition, we have seen that this limit cannot be pushed forever, because of a configuration limitation of the platform and also to avoid being run out of memory space. In consequence in this section we analyze other alternatives to reduce the time spent in the communication phase. In a previous work [27] we introduced an optimization that consisted on parallelizing the data transfer. We explore more deeply this possibility and present a new alternative.

5.1. Parallel-send optimization

Our proposal to parallelize the data transfer operations reduce the time of the communication phase "theoretically" to the time of sending the biggest message and consequently minimize the impact of load imbalance. Figure 12 shows the source code of the optimization.

Parallelizing the cumulative sends performed by each task at each iteration, transforms them in a kind of non-blocking send as each send is executed without having to wait for the completion of previous sends. In addition, the sending of messages is dynamically distributed among the available threads (a chunk of iterations) minimizing the potential load imbalance due to different messages sizes.

In the rest of the paper we refer to this optimization as parallel-send optimization.

DOI: 10.1002/cpe

5.2. Fragmentation of messages

Rather than augmenting the eager-limit, an alternative to avoid switching to rendez-vous protocol is to limit the message sizes. Moreover, when encountering a message size greater than the eager-limit, we split the message into smaller ones such that their sizes fit into the MPI eager message-passing protocol. In the rest of the paper we will refer to this optimization as split.

In other words, we try, if the eager-limit allows it, to turn a not-eager-message into more than one eager-message, providing that the execution time reduction justifies the overhead of splitting and running them in parallel. In this way, we are deferring the rendez-vous protocol to bigger message sizes.

6. EVALUATION

The experimental results from the execution of the different optimizations described in the previous section are now presented.

Evaluations concerning the eager-limit optimizations are presented first. In that sense, we show the results on 10Gigabit interconnection network. The results on Infiniband were already presented in a previous work [27]. Then we show performance results related with data parallelization and fragmentation optimizations.

As we are working with the hybrid implementations, we choose a standard approach: an MPI process for each node (inter-node communication). In this way, we are able to test executions on two interconnection networks. We set the number of threads equal to the maximum number of cores per node, which is 16.

The metrics used to compare the performance for the synthetic benchmarks consist of the speedup with respect to the default version and for the miniFE benchmark, the number of floating point operations done per time unit (gigaflops per second).

In the following we show the experimental results for the executions of the benchmarks using 10Gigabit and Infiniband interconnection networks.

6.1. Experimental platform

Table IV. Execution framework

	Library/tool	Version
Benchmark application	miniFE	2.0
Message passing library	Intel MPI	4.1.1.036
C compiler	gcc (omp)	4.7.2 (omp 3.1)
Operating system	Linux	SuSe Distribution 11 SP2
Interconnection network	Infiniband	Mellanox FDR 10
Interconnection network	10 Gigabit	10 Gigabit Ethernet network
Profiling visualizer	Paraver tool	4.3.5

The machine used to perform the experiments is MareNostrumIII [2], a supercomputer based on Intel SandyBridge processors, iDataPlex Compute Racks, a Linux Operating System. It has 3.056 homogeneous compute nodes (2x Intel SandyBridge-EP E5-2670/1600 20M 8-core at 2.6 GHz) with 2GB per core; and 42 heterogeneous compute nodes (2x Xeon Phi 5110 P) with 4GB per core. The machine has two interconnection networks: a) Infiniband Mellanox FDR10: High bandwidth network usually used by parallel applications communications (MPI) and b) Gigabit Ethernet: 10GbitEthernet network usually used by the General Parallel File System (GPFS).

We use MPI [20] as the programming model to allow communication between nodes. Inside each node we use OpenMP [21] to open parallelism.

Table IV summarizes the execution framework used to perform the evaluations.

DOI: 10.1002/cpe

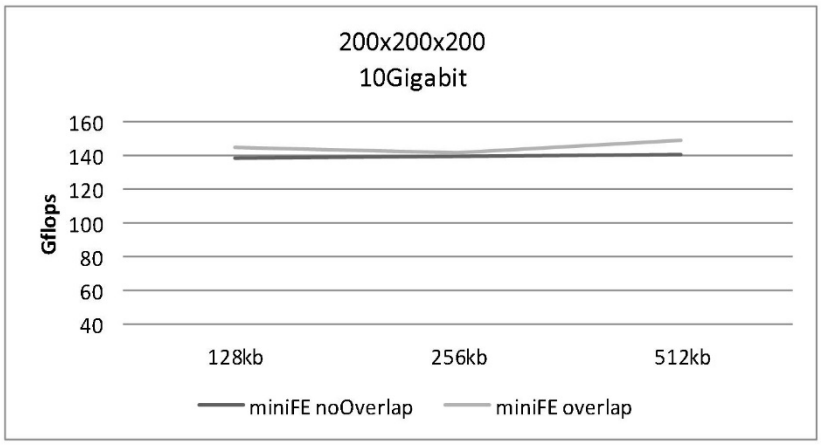

Figure 13. miniFE performance results when adjusting the eager-limit using an input matrix 200x200x200 on 10 Gigabit interconnection network with 64 nodes, varying the eager-limit from 128 Kb to 512 Kb. The biggest message size is about 128 Kb.

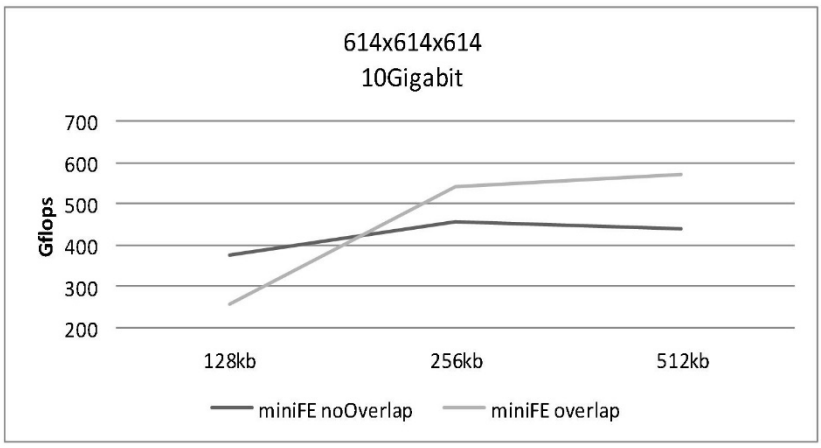

Figure 14. miniFE performance results when adjusting the eager-limit using an input matrix 614x614x614 on 10 Gigabit interconnection network with 64 nodes, varying the eager-limit from 128 Kb to 512 Kb. The biggest message size is about 256 Kb.

6.2. Adjusting the eager-limit to message sizes

Figures 13, 14, and 15 show the performance results in Gigaflops (higher is better) of the execution of the miniFE benchmark with different input matrix sizes and varying the eager-limit when using 10Gigabit interconnection networks and running on 64 nodes (64 MPI tasks, each with 16 OpenMP threads at each node).

Matrix sizes (200x200x200, 614x614x614 and 1048x1048x512) were chosen so they would fit in the memory for that number of nodes, leaving aside memory considerations. The biggest message size for each matrix size is 128 Kb, 256 Kb and 512 Kb respectively.

One may observe that when latencies are higher, as it happens on 10Gigabit, the version with overlap (*miniFEoverlap*) achieves better performance with 200x200x200 and 614x614x614 input matrix size. This improvement is explained because of the overlapping of communication and computation which hides the latencies exhibited when waiting for the completion of the reception of the messages. With bigger input matrix size (1048x1048x512) doesn't perform better because the eager-limit cannot be pushed beyond 256Kb. We confirm that beyond the maximum eager-limit (512 Kb in our platform), tuning is not possible.

When using InfiniBand, on one hand the latency in the completion of the reception of messages is negligible and on the other hand, the MPI protocol of messages does not work as expected. The MPI library switches from eager to rendez-vous under certain circumstances, even though the

Prepared using cpeauth.cls
DOI: 10.1002/cpe

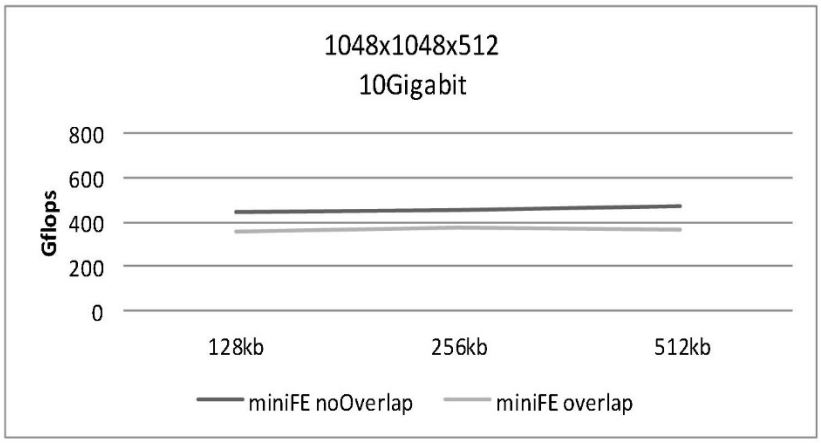

Figure 15. miniFE performance results when adjusting the eager-limit using an input matrix 1048x1048x512 on 10 Gigabit interconnection network with 64 nodes varying the eager-limit from 128 Kb to 512 Kb. The biggest message size is about 512 Kb.

eager threshold is not reached. This happens when there is not enough space to allocate all the receiving messages. In the 1048x1048x512 input matrix case, the synchronized versions show better performance than overlapped as occurs in InfiniBand: the eager protocol cannot be applied, because of big message sizes.

An interesting observation is that the scalability in these examples is so limited that the miniFE overlapped version with eager-limit set at 256 Kb value, using 10Gigabit with input matrix size 614x614x614 has better performance than on InfiniBand. This also happens with the input matrix size 200x200x200.

6.3. Minimizing and balancing the time spent in the communication phase

Then evaluations concerning data parallelization and fragmentation optimizations follow.

We first present a summary of the experimental results for the *noOverlapToy* synthetic benchmark on 10Gigabit (Figure 16) and *overlapToy* synthetic benchmark using 10Gigabit and Infiniband interconnection networks (Figures 17 and 18). The results are presented for the highest communication degree variation (every task send a message to each other task at each loop iteration). Results for *noOverlapToy* on Infiniband are not presented. The communication phase is about 2% so no improvements can be made in that version. Concerning the evaluations using miniFE, we use input matrix size 1048x1048x512, which is the one with messages sizes over the maximum eager-limit.

We are able compare the impact of different latencies at the interconnection network side by doing evaluations on Infiniband and 10Gigabit but leaving the rest of the platform identical (same nodes).

Despite the eager-limit is adjusted we can observe additional performance improvements in the executions of the synthetic benchmarks when adding parallel-send and data fragmentation optimizations. In *noOverlapToy* is noticeable from 2 Mb message sizes, achieving 37% of improvement in the best case (8 Mb message size).

One can observe in the *overlapToy* results, that the performance improvement of the data fragmentation optimization evidences at message sizes close to the eager-limit (128 Kb in Infiniband). This is because what we are really doing is deferring the switching between MPI protocols. In fact, for message sizes close to 8Mb, the performance degrades. This behaviour can be explained due to the increment of the number of messages (e.g. for each 8 Mb message we are generating 16 messages of 512 Kb size) so no way of supporting eager protocol anymore (memory space can run out).

We achieve for the *overlapToy* synthetic benchmark an improvement in the execution time of 27% on 10Gigabit and 43% on Infiniband in average for high communication degree and large message sizes (8 Mb).

DOI: 10.1002/cpe

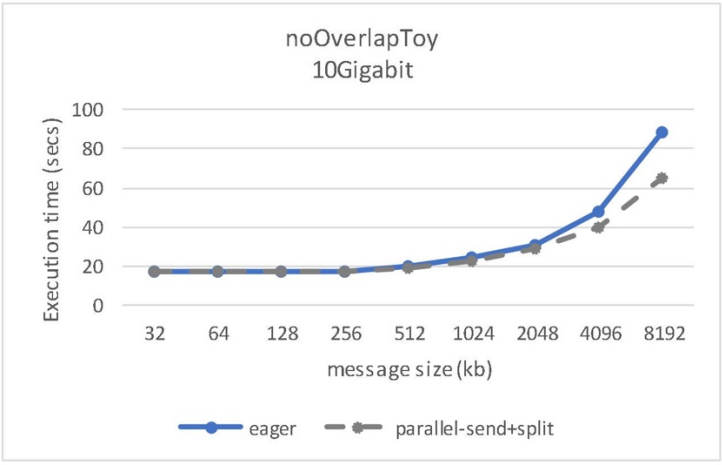

Figure 16. Execution times of *noOverlapToy* with the optimizations applied running on 16 nodes (16 MPI tasks), each with 4 cores (4 OpenMP threads), varying message size and using 10Gigabit interconnection network.

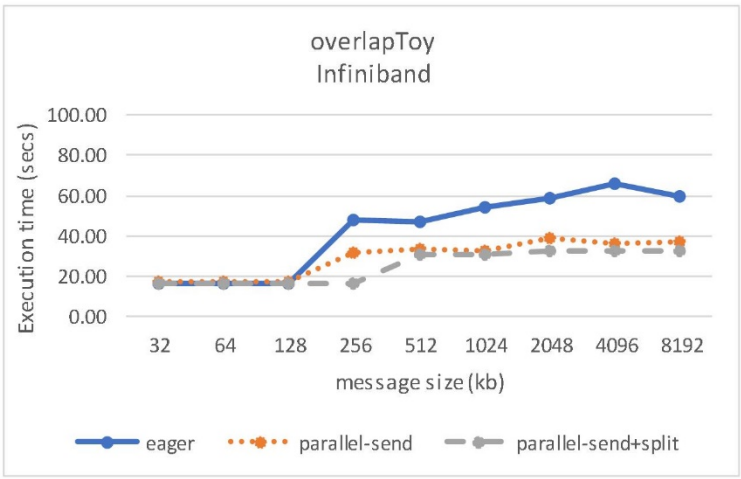

Figure 17. Execution time of *overlapToy* with the optimizations applied running on 16 nodes (16 MPI tasks), each with 4 cores (4 OpenMP threads), varying message size and using Infiniband interconnection network.

The overall impact on the performance of miniFE is shown in Figures 19 and 20 when using InfiniBand and 10Gigabit interconnection networks respectively. The performance of non-overlapped miniFE (*miniFEnoOverlap*) optimized implementation version degrades performance with respect to the version having just eager-limit adjusted at 128 nodes on both interconnection networks. We extended the experiments on InfiniBand to 256 nodes and the results are quite similar to 128 nodes. For small number of nodes evaluated (close to 32) the optimizations has almost no effect with both interconnection networks. However, results on 10Gigabit are quite the opposite. The latencies for this interconnection network justify the different overheads generated by the optimizations. The different scalability between executions with 32 and 64 nodes, and between 64 and 128 nodes for *miniFEoverlap* without optimizations, is due to the message sizes originated by the data distribution with that amount of MPI tasks. For example, when using 32 MPI tasks and 64 MPI tasks, message sizes are around 1 Mb and 512 Kb respectively. Those values are over the allowed eager-limit, which is an important degrading factor in the overlapped versions.

DOI: 10.1002/cpe

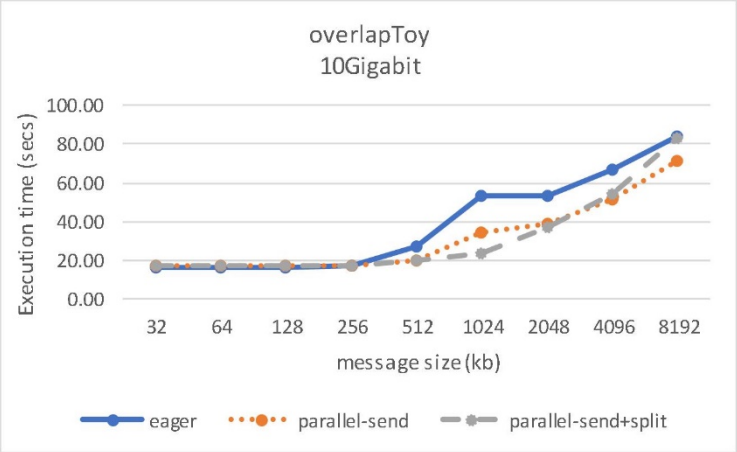

Figure 18. Execution times of *overlapToy* with the optimizations applied running on 16 nodes (16 MPI tasks), each with 4 cores (4 OpenMP threads), varying message size and using 10Gigabit interconnection network.

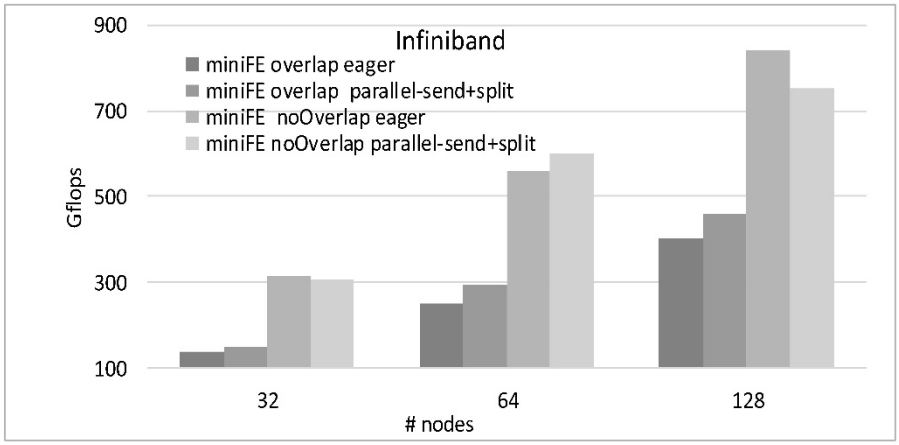

Figure 19. Strong scaling performance results for the miniFE benchmark with non-overlapped and overlapped implementations on 32, 64 and 128 nodes (MPI tasks), each with 16 cores (OpenMP threads), with just eager-limit adjusted (eager) and optimized with parallel-send and data fragmentation (parallel-send+split) running on Infiniband interconnection network.

However, when using 128 nodes, message sizes are around 256Kb, which perfectly fits the eager-limit on that platform.

The best performance increment obtained with the optimizations in the non-overlapped version is about 7% on Infiniband and 11% on 10Gigabit with 64 processes.

The miniFE overlapped optimized implementation shows a performance improvement on both interconnection networks. We obtained in 64 processes an increment about 16% on Infiniband 45% on 10Gigabit.

7. CONCLUSIONS AND FUTURE WORK

In this work, we analyze the load imbalance due to communication on iterative applications by extending a previous study [27] performed on Infiniband Interconnection Network. In order to better analyze the details of the execution, we develop two synthetic benchmarks to represent two types

DOI: 10.1002/cpe

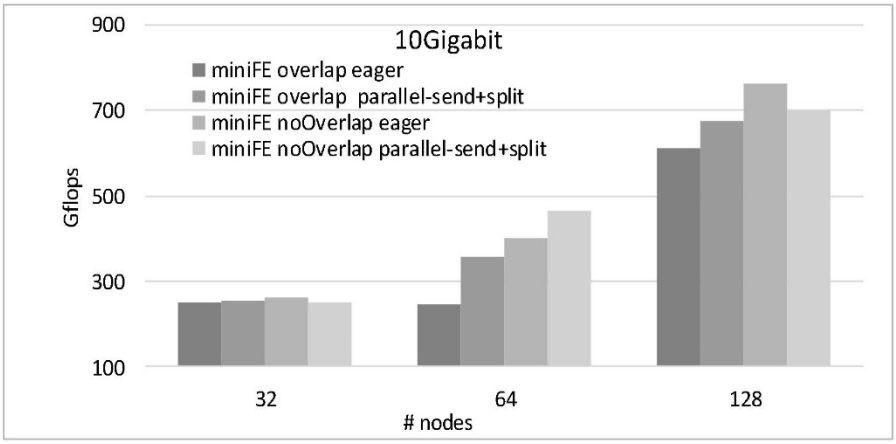

Figure 20. Strong scaling performance results for the miniFE benchmark with non-overlapped and overlapped implementations on 32, 64 and 128 nodes (MPI tasks), each with 16 cores (OpenMP threads), with just eager-limit (eager) adjusted and optimized with parallel-send and message fragmentation (parallel-send+split) running on 10Gigabit interconnection network.

of common point-to-point communication implementations: without communication/computation overlap and with communication/computation overlap. In addition to these benchmarks, we perform experiments on two implementations of the miniFE miniapplication benchmark from the Mantevo project. This application has a dominant kernel composed of a sparse matrix-vector multiplication. All the evaluations are carried out on InfiniBand and 10Gigabit interconnection networks. We observe fast networks with specific optimizations and internal protocol characteristics at different levels, including OS mechanisms, which add variability to the execution behaviour but do not always lead to a better performance.

We analyze the load imbalance generated by the MPI protocol in use, point-to-point communication degree and message sizes used for data distribution. We study several approaches that manage these factors, such as tuning the eager-limit to maintain the eager protocol according to the message size for as long as possible, as well as minimizing the time spent in data distribution. The optimizations proposed impact significantly on the performance of the presented versions of the synthetic benchmarks and miniFE.

The overlapped versions are the most impacted by the eager-limit adjustment. In particular, our performance results show improvements in the synthetic benchmarks up to 3x with message sizes below the maximum allowed eager-limit in 10Gigabit and on Infiniband.

However, the eager-limit cannot be pushed forever, so after reaching the maximum applicable eager-limit other optimizations are analyzed. We have evaluated the effect of diminishing the load imbalance by parallelizing the point-to-point message send operations (proposed in a previous work). We study a new optimization consisting on deferring the eager-limit effect on larger message sizes by fragmenting the messages into sizes that fit this limit. Performance improvements were observed on both interconnection networks, especially on the overlapped versions.

In particular, we obtain up to 37% of improvement with the non-overlapped synthetic benchmark with messages around 8 Mb on 10Gigabit. Similar improvements were obtained for the overlapped versions on Infiniband and 10Gigabit (up to 11% on Infiniband and 45% on 10Gigabit). With respect to miniFE, we obtain performance improvements of the non-overlapped version on Infiniband around 7% and 11% in Gigabit for executions with message sizes around 512Kb. When incrementing the number of nodes, message sizes decrement and the overhead because of opening parallelism and augmenting the number of messages don't justify the improvement.

The performance increment obtained for miniFE is far from the one obtained with the synthetic benchmarks in some cases. The miniFE benchmark has a wider range of message sizes which can mix the different behaviours obtained. In addition, the communication phase in miniFE, has

Prepared using cpeauth.cls
DOI: 10.1002/cpe

a pre-processing of messages which consumes memory and cache misses overhead which was not simulated in the synthetic benchmarks. Anyhow we believe that our results can scale with proper adjustments taking into account all the variables analyzed like the interconnection network, message sizes and communication degree.

We observe that, in some cases, depending on the MPI protocol in use, latency is not a performance factor that differentiates 10Gigabit and Infiniband interconnection networks.

We believe future research on interconnection networks performance should go in the direction of designing new communication protocols having in mind that speed and scalability is a key factor to satisfy exascale necessities. Current communication protocol implementations just take care about messaging overflow by applying buffering as soon as possible. As an example, Intel Omni-Path Fabric [13], recently released as an alternative to the Infiniband Interconnection network and currently in use in MarenostrumIV [3] supercomputer, uses Intel PSM2 Messaging API which specifies by default switches from eager to rendez-vous protocols of 64 Kb for inter node communication and 16 Kb for intra node communications. This strategy (rendez-vous protocol) as we demonstrate in this work, generates bottlenecks in message delivering, limiting the message flow and compromising seriously the performance of the applications.

Future work includes proposals to explore automatic tuning of the eager-limit in more detail as well as scaling of the optimizations analyzed. In this sense, we are currently working on a performance model to estimate the duration of the communication phase, as well as the maximum load imbalance that can be generated.

ACKNOWLEDGMENTS

The authors acknowledge the support of the BSC (Barcelona Supercomputing Centre). We would like to thank the anonymous reviewers for their valuable comments. This work has been supported by the Spanish Ministry of Science and Innovation (contract TIN2015-65316), the Generalitat de Catalunya (2014-SGR-1051) and the the European Commission through the HiPEAC-3 Network of Excellence (FP7/ICT-217068),

REFERENCES

1. T. S. Abdelrahman and G. Liu. Overlap of computation and communication on shared-memory networks-of-workstations. In R. Buyya and C. Szyperski, editors, Cluster Computing, pages 35–45. Nova Science Publishers, Inc., Commack, NY, USA, 2001.
2. Barcelona Supercomputing Center. Marenostrum. https://www.bsc.es/marenostrum/marenostrum.
3. Barcelona Supercomputing Center. Marenostrum IV. https://www.bsc.es/marenostrum/marenostrum/technical-information.
4. Barcelona Supercomputing Center. Paraver: a flexible performance analysis tool. http://www.bsc.es/computer-sciences/performance-tools/paraver/general-overview.
5. D. Bohme, F. Wolf, B. R. De Supinski, M. Schulz, and M. Geimer. Scalable critical-path based performance analysis. In Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pages 1330– 1340. IEEE, 2012.
6. D. Bohme,¨ F. Wolf, and M. Geimer. Characterizing load and communication imbalance in large-scale parallel applications. In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 2538–2541. IEEE, 2012.
7. R. Brightwell and K. Underwood. Evaluation of an eager protocol optimization for mpi. In Recent Advances in Parallel Virtual Machine and Message Passing Interface, pages 327–334. Springer, 2003.
8. A. Danalis, K.-Y. Kim, L. Pollock, and M. Swany. Transformations to parallel codes for communication-computation overlap. In Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC '05, pages 58–, Washington, DC, USA, 2005. IEEE Computer Society.
9. T. Gamblin, B. R. De Supinski, M. Schulz, R. Fowler, and D. A. Reed. Scalable load-balance measurement for spmd codes. In High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for, pages 1–12. IEEE, 2008.
10. HPCG. The High Performance Conjugate Gradients (HPCG) Benchmark. http://hpcg-benchmark.org.
11. HPL. High Performance Linpack benchmark. https://www.top500.org/project/linpack/.
12. IMB Benchmarks. Intel MPI Benchmarks. https://software.intel.com/en-us/articles/ intel-mpi-benchmarks.

13. Intel OPA. Intel Omni-path Fabric. https://www.intel.es/content/www/es/es/ high-performance-computing-fabrics/omni-path-architecture-fabric-overview. html.
14. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput., 20(1):359–392, Dec. 1998.
15. S. Kirmani and P. Raghavan. Scalable parallel graph partitioning. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, page 51. ACM, 2013.
16. B. C. Lee, R. W. Vuduc, J. W. Demmel, and K. A. Yelick. Performance models for evaluation and automatic tuning of symmetric sparse matrix-vector multiply. In Parallel Processing, 2004. ICPP 2004. International Conference on, pages 169–176. IEEE, 2004.
17. J. Liu and D. K. Panda. Implementing efficient and scalable flow control schemes in mpi over infiniband. In Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, pages 183–, April 2004.
18. J. Liu, J. Wu, and D. K. Panda. High performance rdma-based mpi implementation over infiniband. Int. J. Parallel Program., 32(3):167–198, June 2004.
19. MiniFE mantevo suite benchmark. MiniFE from Mantevo suite benchmark. http://mantevo.org/.
20. MPI. Message Passing Interface (MPI) Forum Home Page. http://www.mpi-forum.org/.
21. OpenMP. The OpenMP API specification for parallel programming. http://www.openmp.org/.
22. B. S. Parsons and V. S. Pai. Exploiting process imbalance to improve mpi collective operations in hierarchical systems. In Proceedings of the 29th ACM on International Conference on Supercomputing, pages 57–66. ACM, 2015.
23. F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking, HPCN Europe 1996, pages 493–498, London, UK, UK, 1996. Springer-Verlag.
24. T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in mpi. In Euro-Par 2004 Parallel Processing, pages 173–182. Springer, 2004.
25. M. Small and X. Yuan. Maximizing mpi point-to-point communication performance on rdma-enabled clusters with customized protocols. In Proceedings of the 23rd International Conference on Supercomputing, ICS '09, pages 306–315, New York, NY, USA, 2009. ACM.
26. TOP500. The top 500 list. http://www.top500.org/.
27. G. Utrera, M. Gil, and X. Martorell. Evaluating the performance impact of communication imbalance in sparse matrix-vector multiplication. In 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pages 321–328, March 2015.
28. A. Williams. Multicore and manycore performance studies using a fimnite element miniapplications. SIAM PP10.
29. F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the nas parallel benchmarks. In Supercomputing, ACM/IEEE 1999 Conference, pages 41–41. IEEE, 1999